HIGH-PERFORMANCE HASHING SYSTEM

BACKGROUND

[1] Embodiments of the present invention relate to memory lookup operations using hash functions and, particularly, to such operations that are designed for large scale memories.

[2] "Hashing" generally describes a technique for searching for data within a memory system. Given a set of input data, a hashing function generates an index value. When applied to a memory, the index value should cause requested data to be read therefrom. Unfortunately, depending upon the hash function used, index values may not uniquely identify the requested data. It is possible that a hash function can generate the same index value for two or more unique input values. This is called a "collision." To guard against the possibility of collisions, the index value typically is used as a pointer to a linked list of data. Each element in the linked list typically contains the data being sought (called, the "payload" data herein), a copy of the input data to which it relates and a pointer to the next element in the linked list. In such systems, it becomes necessary to examine each element in the linked list serially until the copy of the input data confirms that responsive data has been found or until the linked list is exhausted.

[3] In those systems described above, the serial examination of each element in the linked list wastes time. It can be particularly disadvantageous in high-performance applications or those involving massive data sets (millions of memory entries or more). Consider, for example, the process of searching an established connection table to support the well-known transmission control protocol (TCP). Given an input tuple that includes an IP source address, an IP destination address, a TCP source port and a TCP destination port, the process must search a memory to retrieve data representative of the connection state. Using a conventional linked list implementation, as the number of active connections grows, the rate of collisions and the length of the linked lists also grow. Hypothetically, if an index hits a linked list with six entries, a system must read each entry in order to detect a match. Because each entry in the list includes a pointer to the next entry in the list, the various entries cannot be read in parallel. Up to six sequential memory reads would be required before it could be determined whether

the input data hit or missed the memory. Thus, the latency problems of such implementations can be severe.

[4] In one well-known TCP implementation, IP source and destination addresses each are represented as 32 bit quantities and TCP source and destination ports are represented as 16 bit quantities. To accommodate all possible variations in these values, a TCP connection table would require 2^{96} entries if implemented without a hash function. A hash function that generates a 32 bit hash value, however, reduces the size of the connection table to 2^{32} entries (about 4.3 million entries). In another TCP implementation, where IP source and destination addresses are represented as 128 bit values, a TCP connection table would require 2^{228} entries. The 32 bit hash value again would reduce the size of the connection table to 2^{32} entries. In this latter implementation, a vastly larger number of unique combinations of input data would map to the same 4.3 million hash values, which raises the collision rate proportionally when compared to the first implementation.

[5] The inventors perceive a need in the art for a high performance hashing algorithm that provides improved performance for large scale memories. They further perceive a need in the art for a hash-based lookup system that avoids the problems of serial reads throughout linked list data structures.

BRIEF DESCRIPTION OF THE DRAWINGS

[6] FIG. 1 is a block diagram of a system 100 according to an embodiment of the present invention.

[7] FIG. 2 is a block diagram of a system 200 according to another embodiment of the present invention.

[8] FIG. 3 is a flow diagram of a method 1000 according to an embodiment of the present invention.

[9] FIGS. 4(a) and 4(b) illustrate a memory space according to an embodiment of the present invention.

[10] FIG. 5 illustrates a method according to an embodiment of the invention.

DETAILED DESCRIPTION

[11] Embodiments of the present invention provide a high-performance, low-latency data retrieval system using hashing. Given a set of input data, the data retrieval system may generate one or more index values and a signature value according to a predetermined hash function. The index values may be applied to respective data arrays to access a data unit therein. The data unit may include signatures previously developed when populating the system. If a signature from the data unit matches the signature generated from the hash function, then the associated index may be applied to a second portion of the respective data array to retrieve the payload data.

[12] FIG. 1 illustrates a data retrieval system 100 according to an embodiment of the present invention. The system 100 may include a hash value generator 110, primary and secondary data arrays 120, 130 and a comparator 140. The hash value generator 110, as its name implies, may apply a predetermined hash function to input data, generating an index value on line 112 and a signature value on line 114. The primary data array 120 may store, for each unique value of an applied address, a data unit 122 storing one or more signature values (S0-S3). The secondary data array 130 may store payload data (e.g., 132, 134) associated with each value of input data.

[13] The system 100 may respond to an external lookup command that seeks payload data in response thereto. The lookup command may include input values that identify the payload data. When an input value is applied to the system 100, the hash function generator 110 may generate the index and signature values 112, 114. The index value 112 may be applied to the primary data array 120 as an address. In response to the index value 112, the primary data array 120 may output a data unit 122 to the comparator 140. The data unit 122 may include a plurality of slots for storage of signature values.

[14] The comparator 140 may compare the signature value on line 114 to any signatures present in the retrieved data unit 122. The comparator 140 may detect not only a match between signatures but also the matching signature's slot position within the data unit 122. In the example shown in FIG. 1, the data unit is illustrated as including four slot positions S0-S3. If the signature at position S2 matches the signature on line 114, the comparator may generate an output on line 142 indicating that a signature match occurred at the third slot position.

[15] The index value 112 and the position value 142 may be input to the secondary data array 130 merged as an address signal. In response, the secondary data array 130 may output a unit of payload data 132 from within the array 130. The payload data can be returned in response to the lookup command.

[16] The foregoing embodiment avoids the latencies inherent in linked list hashing systems noted above. Regardless of the rate of collisions among index values, payload data can be retrieved from a memory system with only two memory reads. Thus, while the given index value still may not represent the input data uniquely, the foregoing embodiments resolve ambiguities through use of a signature in the primary data array. The signature's position helps determine the payload data's position in memory and permits the system to avoid traipsing across multiple entries as in the linked list system.

[17] This embodiment additionally permits system designers to manage the size of data arrays present in the system. Consider an example where a hash function reduces the 2^{228} combinations of IP/TCP input data to a 128 bit hash value. In this example, one may take 32 bits of the hash value as an index value and another 8 bits of the hash value as a signature value. The 32 bit index value leads to a primary array size of 2^{32} (again, about 4.3 million entries). For the purposes of this example, one may choose to provide 8 slots in each entry for storage of a signature value. This would cause the secondary array to have about 34 million entries. By contrast, if one simply were to consider the 32 bit index and the 8 bit signature value as one aggregate index value, it would require an array table having about 1.1 billion entries – about 28 times the size of the primary and secondary arrays combined. Thus, the foregoing embodiment can provide for conservation of memory resources.

[18] It is expected that in practice one may design the primary data array 120 of this embodiment to include a sufficient number of signature positions to accommodate expected collision rates that will occur due to the hash function employed by the hash value generator 110. The higher the rate of collisions, the greater the number of slots there may be.

[19] As explained, the hash value generator 110 may operate according to a hash function. Any of a variety of hash functions may be utilized in various embodiments of the invention. Various hash functions are known and each of them vary in terms of their complexity and the probability of collisions among input values. At a high level, the hash functions generate a hash

value according to an irreducible polynomial. For example, from the 128 bit IP addresses and 16 bit TCP port designators described above, the hash function may generate a 128 bit hash value. The index values and signature values herein simply may be taken from predetermined portions of this 128 value. Although permissible, collision rates can be minimized if the index value and the signature values are taken from non-overlapping portions of the hash value.

[20] Depending upon the hash function used, it may be possible that collisions will occur among signature values. In another embodiment, shown in phantom in FIG. 1, signature collisions may be resolved by storing payload data and its associated input value together in an entry 136 of the secondary data array 130. When data is read out of the secondary data array 130, a second comparator 150 may compare the received input data with the input data that is output from the selected entry 136. If they match, then the payload data from the entry 136 may be returned in response to the lookup command. If not, then the data was selected due to an errant collision of the hash function and it probably is not responsive to the lookup command.

[21] FIG. 2 illustrates a data retrieval system 200 according to another embodiment of the present invention. In this embodiment, the system may include multiple independent sets of primary and secondary data arrays. Two sets (set 1 and set 2) are shown in the example of FIG. 2 but the number may be increased as needed to accommodate any expected rate of collisions between the various index values and signature values.

[22] The system 200 may include a hash value generator 210, a pair of primary data arrays 220, 230 and a pair of secondary data arrays 240, 250. In this embodiment, the hash value generator may apply a hash function to input data to generate a hash value therefrom. A first portion of the hash value may be used as an index (index1) to be applied to the first primary data array 220. A second portion of the hash value may be used as a second index (index2) to be applied to the second primary data array 230. A third portion of the hash value may be selected for use as the signature.

[23] Responsive to the index1 value, the first primary data array 220 may output a data unit that includes a plurality of signatures provided in predetermined slot positions therein. The signatures from the first primary data array 220 each may be compared to the signature from the hash value generator 210. On a match, the index1 value and a position value indicating the

slot position of the matching signature may be applied to the first secondary data array 240. These inputs may cause the first secondary data array 240 to output a data value therefrom. The output of the first secondary data array 240 may be responsive to the lookup command.

[24] According to an embodiment, complementary processes may occur in the additional sets of primary and secondary data arrays (e.g., 230, 250). The index2 value, when applied to the second primary data array 230 may cause data to be output therefrom. The data may include a plurality of previously-stored signature values at predetermined positions therein. A comparison may be made of the output signature values and the signature value output from the hash value generator 210. If a match occurs, the index2 value and the matching signature's position may be applied to the second secondary data array 250. The second secondary data array 250 may output data that is responsive to the lookup command.

[25] The embodiment of FIG. 2 further reduces the likelihood of collisions among output data. As in the embodiment of FIG. 1, the use of an index value (say, index) and a signature value reduces the likelihood that the system will permit multiple sets of input data to refer to the same entry in the primary data array. Additionally, the FIG. 2 embodiment provides additional protection against collisions through use of multiple index values (index1, index2) with a signature value.

[26] According to an embodiment of the present invention, shown in FIG. 2 in phantom, the system 200 may include an additional layer of comparators 260, 270 coupled to outputs of the secondary data arrays 240, 250. In this embodiment, entries of the secondary data arrays may include both payload data and the input data to which the payload data corresponds. The comparators 260, 270 may determine if the data input to the hash value generator 210 matches the values output from the respective secondary data arrays 240, 250. If so, then it is confirmed that the payload data is responsive to the input data. Otherwise, the payload data can be considered non-responsive.

[27] FIG. 3 is a flow diagram of a method 1000 according to an embodiment of the present invention. When new input data is available, a hash function may operate on the input data (box 1010). An index value and signature value may be selected from the hash value obtained thereby. The index value may be used to retrieve a first data unit from a data array (box 1020). Thereafter, the signature value may be compared to signature values contained within

the data unit to determine whether there is a match (boxes 1030, 1040). If a match occurs, then a second data unit may be retrieved from a memory using the index value and the matching signature's position within the retrieved data unit as an address (box 1050). The second data unit may contain data responsive to the lookup command (box 1060).

[28] If none of the signatures from the first data unit match the newly generated signature, then the system returns a response to the lookup command indicating that the requested data is not present in the memory (box 1070).

[29] In an alternate embodiment, when the second data unit is retrieved from memory, the method may compare the input data to an input data field present in the second data unit and determine if there is a match between them (box 1080). If so, the method may proceed to box 1060, using payload data stored in the second data unit as a response to the lookup command. If not, the method may proceed to box 1080 and return an indication that the lookup command missed the memory (box 1070).

[30] In embodiments involving multiple index values obtained from the hash function, for example, index1 and index2 from FIG. 2, the method 1000 may perform the operations illustrated in boxes 1020-1080 independently for each index value. Bounding box 1090 is provided in FIG. 3 to illustrate this functionality.

[31] In certain applications, if a lookup command misses the memory system, a new entry may be allocated to the input data of the lookup command. For example, in a TCP application, if input data does not refer to an active connection, then a new connection will be established. Thus, an embodiment permits a new entry to be allocated to the memory system on a miss. Allocation of a new embodiment can cause the signature value to be stored in an unoccupied slot in a primary data array (box 1100) and payload data to be stored in an entry of a corresponding secondary data array at a position identified by the index and the slot now occupied by the generated signature (box 1110).

[32] The foregoing description has explained the operation of the present invention in the context of discrete data arrays. In an embodiment, each of these arrays may be distributed throughout a common memory system as is shown in FIG. 4. FIG. 4 (a) illustrates primary and secondary data arrays 410, 420 in a memory space, where each data array is identifiable

through a base memory address. Accessing data units from a primary data array 410, in this embodiment, can be as simple as using the index value as an offset from a first base address. Accessing data units from a secondary data array 420 in this embodiment may occur by using the index value and the position value as an offset from the secondary data array's base address. Of course, there is no requirement that primary and secondary data arrays 410, 420 be provided in continuous spaces in memory or that they be adjacent to one another as shown in FIG. 4(a). In one embodiment, the smaller primary data array 410 may be provided in an SRAM memory and the secondary data array 420 may be provided in a DRAM memory. The principles of the foregoing embodiments can be employed cooperatively with other memory management schemes as may be desired.

[33] In another embodiment, illustrated in FIG. 4(b), stored signatures and corresponding payload data may be stored as contiguous units 430, 440 in memory. In this system, signatures may be stored at memory locations offset from a base address by an amount N*L*index, where N represents the number of signatures stored per primary array entry (N=4 in the example of FIG. 4(b)) and L represents the length of the signature field in bytes. Associated payload data may be stored in memory locations adjacent to the stored signatures. If a signature match occurs, the payload portions may be offset from the primary array entry by the slot position of the matching signature (e.g., (N*L*index)+1, (N*L*index)+2, etc.).

[34] FIG. 5 is a flow diagram, according to an embodiment, of a method 2000 that selects one of multiple sets of data arrays to store new payload data. As noted, during a lookup operation, various index values may cause a data unit to be retrieved from each of the primary data arrays. According to an embodiment, the method may count the number of unoccupied slots in each of these data units (box 2010); the unoccupied slots are available to store a new signature value. The method also may determine how many data units generated a signature match in any position during the lookup operation (box 2020). If none of the data units caused a match with the generated signature value, then the method may determine which of the data units has the greatest number of empty slots (box 2030). The method may cause the signature value that was generated during the lookup operation to be stored in an empty slot of the data unit in the corresponding primary data array (box 2040). The method also may cause payload data to be stored in a position of an associated secondary data array at a location dictated by the index value and the position of the now-occupied slot in the primary data array (box 2050).

The operations described at boxes 2030, 2040 and 2050 also may be employed if signature matches occurred for all primary data arrays in the system.

[35] If some of the data units caused a match, then the payload data may be stored in one of the sets that does not cause a match (box 2060). The signature value may be stored in an empty slot of the primary array and the payload data may be stored in entry of the secondary array that corresponds to the index and the position of the now-occupied slot (boxes 2070, 2080). If there are multiple data items that did not cause a match with the signature then, of these, the set that corresponds to the data unit having the greatest number of empty slots may be selected for use in a manner consistent with box 2030.

[36] Several embodiments of the present invention are specifically illustrated and described herein. However, it will be appreciated that modifications and variations of the present invention are covered by the above teachings and within the purview of the appended claims without departing from the spirit and intended scope of the invention.